2011

A performance analysis of Xen and KVM hypervisors for hosting the Xen Worlds Project

Charles David Graziano
Iowa State University

A performance analysis of Xen and KVM hypervisors for hosting the Xen Worlds Project

by

Charles David Graziano

A thesis submitted to the graduate faculty
in partial fulfillment of the requirements for the degree of
MASTER OF SCIENCE

Co-majors: Information Assurance
Computer Engineering

Program of Study Committee:
Thomas E. Daniels, Major Professor
Douglas Jacobson
Joseph Zambreno

Iowa State University
Ames, Iowa
2011

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Virtualization of the operating system has become an important consideration in the cloud, corporate data center, and academia. With the multitude of available virtualization platforms, careful consideration is needed for selecting the right solution for a specific virtualization application. This research focuses on an analysis of two open source virtualization platforms or hypervisors: Xen and Kernel-based Virtual Machine (KVM). Evaluations are conducted on the basis of overall performance and throughput, virtual machine performance isolation and scalability for use as a host for the Xen Worlds Project. In doing so, the existing Xen Worlds Project infrastructure, middleware, and virtual machines are migrated to KVM to provide a test bed for the evaluation using a benchmark suite and a set of scripts to simulate user behavior.

CHAPTER 1. OVERVIEW

1.1 Introduction

In recent years virtualization has gained popularity in many different areas such as server consolidation, information security and cloud computing. This is largely due to an increase in hardware performance of about ten fold in the past decade and the goal to reduce capital and operational costs within the data center. [4] Virtualization can help IT managers fully maximize the potential of their hardware investment by increasing the number of applications and operating systems running on a single physical server. Similar goals exist within academia.

Several projects exist that make use of virtualization in one form or another to provide a virtual lab environment of varying levels of sophistication and scalability to students. The primary goal for all of them; to reduce the physical hardware necessary to administrate and maintain computer security lab assignments. When developing such a platform for academia several factors become must also be considered. Things such as the hardware requirement, guest operating systems, type of assignments, and how it is to be administrated and accessed become very important. In this day in age the method of providing virtualization is key. There are a multitude of platforms and options to consider each having their own advantages.

In this thesis the performance characteristics of two different virtualization platforms: Xen and Kernel-based Virtual Machine (KVM) will be analyzed specifically for their use as a virtualization platform for the Xen Worlds Project. [1] Chapter 2 will give background information on virtualization and Chapter 3 will explain the two platforms be assessed. Chapters 4 and 5 will give a brief background on the Xen Worlds Project and platform, feature benefits of KVM followed by the process undertaken to migrate to KVM. Multiple performance evaluations comparing the two hypervisors are performed in Chapter 6. Chapters 7 outlines Xen

Worlds specific features enhancements provided by KVM followed by possible future directions and conclusion in Chapter 8.

CHAPTER 2. WHAT IS VIRTUALIZATION

Virtualization can trace its roots back to the mainframes of the 1960's and 1970's. While the definition of virtualization has evolved since then initially it was considered a method of logically dividing mainframes to allow for multiple applications to run simultaneously. Prior to this time, systems could only execute a single application at a time. Due to the high cost of these mainframe systems many organizations found it difficult to justify the investment for a system which could only due one thing at a time. [10] This changed with the advent of the VM mainframe from IBM and has continued to evolved since.

With the advent of relatively inexpensive x86 based systems in the 1990s and the adoption of Windows and Linux operating systems lead to a shift away from mainframe based applications to the more distributed client-server based applications. It was now commonplace for these systems to run multiple applications simultaneously although not in the sense that it was done in the mainframes. It wasn't until the early 2000s where virtualization began to evolve as x86 based systems became more powerful.

As corporate data centers began to grow so did the cost of supporting the high number of systems. Especially as applications were generally dedicated their own server to avoid conflicts with other applications. This practice caused a waste in computing resources as the average utilization for many systems was only 10% to 15% of their possible capacity. [10] It's at this point many companies started looking at virtualization for a solution. Unfortunately, the now common x86 systems were not designed with virtualization support in mind which caused challenges when attempting to handle some of the privileged instructions in the x86 architecture.

Throughout the 2000's several solutions were developed by companies and the open-source community which were able to provide a method of handeling the privileged instructions in one

way or another and allow for multiple operating system to run simultaneously on a single x86 based system. This lead to a change in the definition of virtualization to the all encompassing:

> Virtualization is a framework or methodology of dividing the resources of a computer into multiple execution environments, by applying one or more concepts or technologies such as hardware and software partitioning, time-sharing, partial or complete machine simulation, emulation, quality of service, and many others. [15]

2.1 Types of x86 Virtualization

As x86 virtualization continues to develop many are looking at virtualization on more than just servers. Virtualization at the desktop and applications virtualization are both areas continuing to grow.

2.1.1 Application Virtualization

Application virtualization brings the virtualization up a layer from the operating system making the applications independent from the underlying operating system. This becomes useful when using legacy applications on new operating systems or when crossing flavors of operating systems. When the virtualized application is executed it is fooled into thinking it is running on the original supported operating system. Typically this requires that a small runtime environment be installed on the system wishing to run the virtual application. This approach offers several advantages compared to running the application natively including:

- Allowing for multiple versions of applications to be used on the same system at the same time.
- Simplifying compatibility issues with other applications allowing potentially incompatible applications to coexist on the same system.
- Simplifying application deployment and upgrades helping maintain consistency throughout an organization.

- Increasing security by removing the need for end users to have Administrator access on their workstation.

2.1.1.1 Application streaming

This sub-type of application virtualization can allow for the desktop PC to be completely removed and replaced with a thin client with network access. Instead of running applications locally, either native or virtual, this practice allows for applications to be executed on servers within the datacenter and enabling users to interface with them from their workstations and many times outside the organization via the internet. This allows for better allocation of fiscal and computational resources as end user workstations can be replaced with an relatively inexpensive thin clients. The savings can then be used within the data center on shared server grade infrastructure to host the applications.

2.1.1.2 Virtual Desktop

Similar to application streaming, virtual desktop allows for the full desktop operating system to be virtualized within the data center and presented to the end user via a thin client. Unlike application streaming the user can be given a virtual desktop instance which they can customize to suit their needs but still have it running on the shared infrastructure within the datacenter. The users are able to interact with the virtual desktop the same way they do with a traditional desktop. Unlike a traditional desktop a virtual desktop can be easily made accessible from anywhere via the internet.

2.1.2 Operating System Virtualization

The more widely thought of, operating system virtualization, allows for multiple operating system instances to run simultaneously on a single physical machine. While it is the base for Virtual Desktops, it was initially designed and primarily used within the data center to virtualizes servers to increase the physical hardware's utilization. It is also used on workstations to allow users to run multiple operating systems on their machines. This can be useful in development environments where software can be tested on multiple operating system instances, in

operating system evaluations, or when a user occasionally needs access to a alternate operating system. It is this type of virtualization for which Xen and KVM are used and is being evaluated in this study.

2.2 Components of Virtualization

2.2.1 Hypervisor

The hypervisor, also known as a Virtual Machine Monitor (VMM) is the software layer which enables virtualization. It is responsible for creating the virtual environment on which the guest virtual machines operate. It supervises the guest systems and makes sure resources are allocated to the guests as necessary. Generally hypervisors are classified into one of two categories; Type 1 and Type 2.

2.2.1.1 Type 1

The Type 1 hypervisor is considered a native or bare metal hypervisor. This type of hypervisor is the lowest level hypervisors, running directly on the host hardware. It is responsible for allocation of all resources (disk, memory, CPU, and peripherals) to its guests. These hypervisors typically have a rather small footprint and do not, themselves, require extensive resources. Occasionally, they have very limited driver databases limiting the hardware on which they can be installed. Some also require a privileged guest virtual machine, known as a Domain-0 or Dom0, to provide access to the management and control interface to the hypervisor itself. The Type 1 hypervisor is what is most typically seen in the server virtualization environment.

2.2.1.2 Type 2

The Type 2 hypervisor requires a full host operating system in order to operate. That is to say that it is installed on top of the host operating system. This has some advantages in that it typically has fewer hardware/driver issues as the host operating system is responsible for interfacing with the hardware. Type 2 hypervisor can be used for application portability such

is the case for the Java Virtual Machine (JVM) or to run operating systems. The downside being additional overhead can cause a hit on performance compared to Type 1 hypervisors.

2.2.2 Guest

The guest, whether an application or an operating system, is what is being virtualized on top of the hypervisor. In some cases an operating system is able to run native and unaltered on the hypervisor necessarily knowing it is virtualized. Other times, depending on the hypervisor, the guest operating systems need special drivers or to be specifically designed to be run on the hypervisor.

2.3 Why Virtualize

Virtualization offers many advantages over traditional infrastructure in a wide variety of applications. Apart from the increasing host utilization through server consolidation it also offers many side benefits as well. These range from reducing fiscal costs to the ease of managing and deploying additioanls systems in a virtual infrastructure. Many virtualization products allow for a specific guest to be easily duplicated which can be useful for testing how software updates or modifications can affect a given system without affecting production systems. Online or offline migration is useful in the event that a specific physical host needs to be upgraded or replaced. Migration helps to reduce or eliminate downtime as the virtual machines can be moved temporarily or permanently to a different host transparent to the guest operating system. Disaster recovery efforts can be greatly simplified through the use of virtualization. In the event of a disaster, backups of the virtual machines can be easily brought up in a new location on different hardware helping to minimize downtime.

Even with all of the advantages there are still some downsides to virtualization. For instance, virtual machines must share resources with other virtual machines on the same physical system. Applications with high disk or CPU utilization do not necessarily perform well on a system where they must share resources. This problem can be aided by only running a single one of these high utilization applications per physical host. This approach allows for the same portability advantages of the other virtual systems while minimizing the performance hit. This

largest problem comes from applications which need access to special physical peripherals. While some virtual platforms may have methods of passing physical peripherals to its virtual machines but then the guest will be restricted to operating on a that specific host.

2.3.1 Infrastructure as a Service

As system virtualization becomes more popular and powerful several providers have built virtual 'cloud' environments within their own data centers and lease space to other organizations. This practice has become known as Infrastructure as a Service (IaaS). Instead of going through the cost of building their own data centers, IaaS allows organizations to lease compute power from one or more data centers built by one or more providers typically on a per-use basis. This model removes the cost of housing, maintaining, and running the physical hardware from the end user and places it on the cloud provider. It does however, allow an organization to quickly scale and reduce it usage as their situation requires.

IaaS has its disadvantages primarily in terms of privacy and security. In order to make use of IaaS company data must be stored or at least accessible from that cloud. This means the data must leave the organization and becomes an important consideration if the organization is subject to one or more data protection laws or regulations. When choosing a IaaS provider, the organization should verify the provider has the necessary steps in place to comply with the necessary laws and regulations.

CHAPTER 3. THE HYPERVISORS

3.1 Xen

The Xen hypervisor is an established Type 1 or bare metal hypervisor which runs directly on the host hardware. Xen is well known for its use of paravirtualization and near-native performance. [5] The Xen hypervisor is managed by a specific privileged guest running on the hypervisor known as Domain-0 or Dom0. Dom0 is a specially modified Linux kernel which is started by the Xen hypervisor during initial system start-up. It is responsible for managing all the aspects of the other unprivileged virtual machine or Domain-Us (DomU) that are also running on the hypervisor: see Figure 3.1.

There can be one or more DomU guests running but none of them have direct access to the physical hardware but instead are required to make CPU, I/O and disk requests to the Xen hypervisor. Originally, DomU guests required a modified kernel ported specifically to run on the Xen hypervisor. This practice, known as paravirtualization (PV), has the advantage that the guests are aware that they are running on a hypervisor and thus do not require the overhead associated with emulating hardware in order to operate. The normally privileged kernel instructions, such as memory and CPU management, are converted to "hypercalls" in a paravirtualized environment and easily executed by the hypervisor instead. PV allows for guests to operate on host systems which do not have CPU hardware virtualization extensions. [8] PV Xen DomU support is now included within the upstream Linux kernel, however, Dom0 support is not. This makes it much easier for linux to be used as a guest than a host system.

While initially only supporting PV Xen now supports the use of new virtualization processor extensions added to the x86 architecture. Known by Xen as Hardware Virtual Machine (HVM) this allows for unmodified guest operating systems to be virtualized on Xen hypervisor.

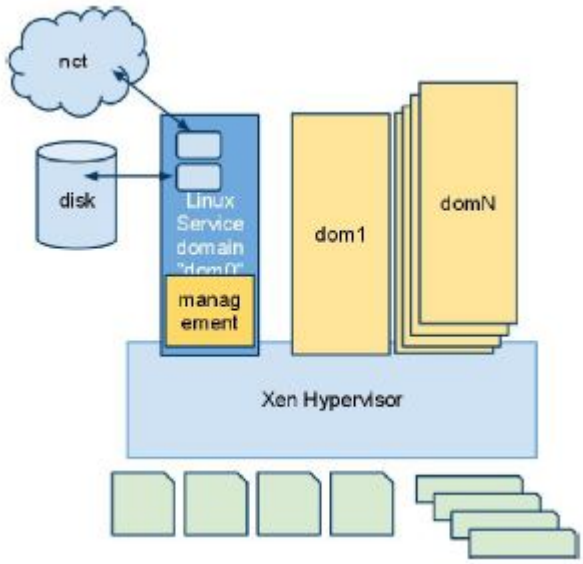

Figure 3.1 Depiction of the Xen hypervisor and how dom0 and domUs are layered on top of the hypervisor.

However, HVM requires processors which specifically support the hardware virtualization extensions (Intel VT or AMD-V). The virtualization extensions allow for many of the privileged kernel instructions which in PV were converted to "hypercalls" to be handled by the hardware using the trap-and-emulate technique. This is similar to the classic Full Virtualization [8] but instead of being done by software it is done within the hardware to increase performance.

3.2 Kernel-based Virtual Machine

Kernel-based Virtual Machine (KVM) is a relatively new hypervisor that has gained momentum and popularity in the past few years. While not as well known as Xen, it has the distinction of having both host and guest support incorporated into the upstream Linux 2.6.20 kernel released in January 2007. KVM is implemented as a kernel module which, when loaded, converts the kernel into a bare metal hypervisor. [7] KVM was designed and implemented after the development and release of hardware assisted virtualization extensions and has been optimized to take full advantage of these extension rather than building them in as an afterthought. Due to this, KVM requires that the Intel VT or AMD-V extensions be present and enabled on the host system. By converting the Linux kernel into the hypervisor KVM developers are

able to take advantage of and build on many components which are already present within the kernel such as the memory manager and scheduler instead of building them from scratch.

Being a loadable kernel module and part of the upstream kernel patching and updating KVM is much easier when compared to Xen. The split architecture of Xen requires maintenance of both the Xen hypervisor and Dom0. Additionally, any changes or enhancements to the upstream kernel must be back-ported to Xen Dom0s increasing the burden on vendors.

KVM is architected such that, from the view of the host, each virtual machine is a standard Linux process, see Figure 3.2; managed, scheduled and secured as a standard Linux process. A modified version of QEMU is used to provide emulation for devices such as the BIOS, PCI bus, USB bus, disk controllers and network cards.

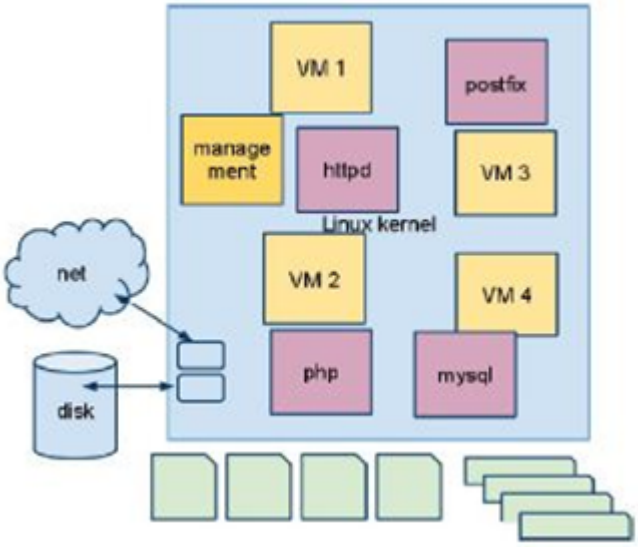

Figure 3.2 Depiction of how virtual machines appear as a regular system process to the KVM hypervisor.

While not required, KVM can be used with standard paravirtualized VirtIO drivers installed on the guest. VirtIO is a standard framework allowing for guests to be more easily transferred between hypervisor platforms supporting VirtIO. This allows for increased I/O performance for network and block devices when compared to regular emulated devices. VirtIO drivers are included in Linux kernels after 2.6.25 and are also available for Microsoft Windows guests as well.

CHAPTER 4. XEN WORLDS BACKGROUND

Xen Worlds is an environment developed at Iowa State University for use by graduate and undergraduate classes. Its goal is to provide students with a network of virtual machines (called a Xen World) on which to perform class assignments and lab work. It is accessible on a 24/7 basis by both on-campus and off-campus students via SSH. Each Xen World is segregated from every other Xen World as well as the outside world (internet). This allows students to be given root access to their Xen Worlds while preventing access to other students Worlds and the hypervisor/Dom0.

Xen Worlds makes use of several open source applications in order to provide the lab environment to students. The current environment uses the Xen hypervisor included with Red Hat Enterprise Linux 5.3 (RHEL) as its virtualization platform. Custom Xen Worlds Middleware was written to handle management and deployment of Xen Worlds assignment. Additionally, Pdmenu, accessed via SSH, is used to provide students with an easy to use interface for accessing and managing their Xen Worlds virtual machines. The developers have created several information assurance related assignments using RHEL 5.3 paravirtualized on the Xen hypervisor. [1]

4.1 Why KVM

The new release of Redhat Enterprise Linux, RHEL 6, no longer supports running as a Xen dom0. [13] Instead Redhat has decided to migrate to KVM as its supported virtualization platform in RHEL6 [7]. Many different performance evaluations have been performed on various virtualization platforms, including Xen and KVM, using a variety of different evaluation criteria. The transition to KVM in RHEL 6 led to the interest in evaluating KVM's suitability as a host

for the Xen Worlds Project environment as well as options for further development and feature enhancements.

4.2 Major Feature differences

Several features exist within KVM which either do not exist in Xen or were not implemented in the original Xen Worlds environment. For example, the method in which memory is allocated to the virtual machines is very different. In Xen, if a guest is provisioned 2GB of RAM when that guest is started 2GB of RAM is immediately allocated to that guest and cannot be used by any other guest. KVM does soft provisioning where memory is allocated to the guest when it needs it up to the provisioned amount. This can allow for memory overcommit, where more memory is assigned to virtual machines than what is available on the system.

KVM has a couple of ways of dealing with memory overcommit. Since a guest in KVM is a regular Linux process some memory pages can swapped out to disk. In this case the host picks the memory pages and writes them to disk. This can have obvious performance consequences when the guest needs to access memory the host will need to read the memory which has been swapped to the disk. Through the use of VirtIO drivers, the KVM host can work with the guests to request that the guests shrink their cache to help free memory. This approach, known as ballooning, requires that the guests cooperate with the hosts request to shrink their cache.

KVM also has the ability to do Kernel Samepage Merging (KSM). KSM works by merging identical memory pages from multiple guests into a single read-only memory region removing duplicate copies. In the event one of the guests attempts to write to one of these pages, a writeable copy is created for that guest to modify. The KSM approach works best when guests are running the same operating system and similar applications in order to maximize the number of identical memory pages. [6]

Both Xen and KVM support multiple methods of storing guest's virtual disks. Xen Worlds uses a flat raw image file for each guest. When a Xen Worlds assignment is deployed, this requires a full copy be created for each guest. This is time consuming and requires significant disk space. KVM is able to utilize a copy-on-write image file format. This allows deployed guests to utilize a base image as a read-only backing file while writing changes to a guest

specific write enabled overlay file. This practice drastically reduces the required disk space and has the added bonus of supporting guest snapshots allowing users to easily revert their guests to a previous state. While newer versions of Xen do support copy-on-write image files, support at the time of Xen Worlds original development support was very lacking. [3]

CHAPTER 5. MIGRATING TO KVM

5.1 Migrating Middleware

The Xen Worlds Middleware is a program written in Python which manages the deployment of Xen Worlds assignments. In order to be compatible with KVM, the middleware had to be modified to replace the existing Xen hypervisor specific commands with their KVM equivalent. Multiple modifications were also made in the way assignments were deployed. First, the middleware was adapted to utilize copy-on-write images for the guest virtual machines.

Additionally the method in which individual virtual machines are customized for each user and the lab was altered. Previously, virtual machine specific commands such as setting the hostname and IP addresses were built into an Expect script and executed on each guest during the lab deployment. This required that each virtual machine be started during the deployment phase in order for the Expect script to be executed. Since all current lab virtual machines are built on a Red Hat environment this approach was changed to use the libGuestFish Python library to directly access the copy-on-write image files. Instead of using an Expect script the same commands are placed in the rc.local file to be executed the first time the virtual machine is started. After the customizing the system the extra commands are removed from the rc.local file.

The last feature modification to the middleware was the addition of snapshots, a feature of the copy-on-write image files. After configuring the virtual machine a snapshot is created. Previously, if a user messed up in a way that could not be easily fixed they had to contact the instructor to re-deploy their virtual machine. This snapshot feature allows the user to easily revert their virtual machines back themselves.

5.2 Migrating User Interface Pdmenu

Xen Worlds uses Pdmenu to provide users with a simple menu driven method of accessing their Worlds as seen in Figure 5.1. The configuration file driving Pdmenu was updated to support KVM and it was enhanced to enable the user driven "revert to snapshot" feature as well as on the fly generation of the menu. Previously, the Pdmenu configuration had to be manually modified by the instructor in order to allow users to access new worlds. A side effect of this is that if multiple assignments were deployed to different subsets of users, every user would be given the same menu interface. Pdmenu has been updated to generate a menu on the fly automatically customized to each user. This modification makes it so that each user can only see Worlds that have been assigned to him/her helping to alleviate confusion on the side of the user and instructor.

Figure 5.1 Screenshot of the Pdmenu interface

5.3 Migrating Labs VMs

In order to make the KVM implementation of Xen Worlds usable the various labs needed to be migrated to KVM as well. Two different approaches were taken to transfer the existing labs to the KVM environment; rebuild the lab and a virtual-2-virtual (v2v) conversion.

The easiest method of transferring the lab virtual machines to KVM was to rebuild them from scratch as a KVM virtual machine. This also allowed for simple installation of KVM

specific packages such as the VirtIO drivers and the acpid daemon to enhance disk and network throughput and allow for graceful OS shutdowns, respectively. Additionally, lab specific application and source files could be transferred and compiled on the new virtual machine via the same KickStart file that was executing the install, further simplifying installation. The newly rebuilt virtual machines were also upgraded from RHEL 5.3 on Xen to RHEL 6, which is supposed to be more highly optimized to run well on KVM [6].

Unfortunately, not all labs could be easily transferred to KVM. The PirateSoft Lab, where students "traverse a fictitious company network (PirateSoft) and find the various parts of a secret project currently under development" [1] could not be rebuilt on KVM. This lab consists of a network of seventeen "company" machines and student machines. The student machines were rebuilt to RHEL 6 however the "company" machines were too specifically configured with special "flags" in various locations on various machine to make rebuilding an infeasible option. Instead a v2v utility available from Red Hat [6] was used to migrate virtual machines from Xen to KVM was used. The program was largely successful however the boot loader (grub) and various other packages needed to be manually updated after the conversion to support some of KVM's features.

CHAPTER 6. EVALUATING KVM'S PERFORMANCE

Several different tests were performed on both the Xen and KVM environments in order to evaluate their performance differences and KVM's suitability to host the Xen Worlds environment. The evaluations were done using two Dell PowerEdge servers each with two dual-core Xeon processors running at 1.6GHz with 32GB of memory and disks configured in a RAID-5 array. One server is running the current Xen Worlds environment consisting of RHEL 5.3 64bit running Xen hypervisor version 3.1.2 while the other server is running RHEL 6 64bit with KVM. Two different classes of performance tests were executed. One to test the overall performance of the hypervisors under various levels of load and another to examine how increasing the number of deployed Worlds on a system can affect the interactive responsiveness of other Worlds.

6.1 Phoronix Test Suite

The Phoronix Test Suite (PTS) [12] is an open source benchmark platform which was used to benchmark and compare various system attributes. Of the multitude of tests and profiles available five were chosen to measure various performance attributes important in a virtual environment. These attributes include CPU usage, disk access rate, memory access rate, and how well the hypervisor is able to handle high loads distributed across several virtual machines running simultaneously.

The five PTS benchmarks which were selected are:

- Apache benchmark: measures how many requests per second a given system can sustain when carrying out 500,000 requests with 100 requests being carried out concurrently
- John the Ripper: benchmarks of how many Blowfish computations can be done per second

- OpenSSL: measures the RSA 4096-bit encryption and decryption performance
- Stream: benchmarks system RAM read and write performance
- Unpack: measures the amount of time it takes to extract content from a .tar.bz2 archive

Each of these benchmarks was executed on virtual machines running on both the Xen and KVM hypervisors. Virtual machines running the Xen hypervisor were running RHEL 5.3 and virtual machines running on the KVM hypervisor were running RHEL 6. Both had fresh images created solely for running these benchmarks and each virtual machine was allocated 2GB of RAM. Additionally, no other virtual machines were running on the hosts other than the machines actively running the benchmarks. Since Xen supports both the paravirtulization (PV) and hardware-assisted virtualization (HVM) each of these operating modes were tested to aid in the comparison against KVM.

In order to insure repeatability and ease of execution a stack scripts was created which would, in part, insure that the benchmarks ran concurrently on each virtual machine being tested, making use of the Linux "at" command, and would automatically upload the results to an FTP repository upon completion. The scripts also insured that all running benchmarks completed and uploaded their results prior to setting up and executing the next test.

6.2 Xen and KVM with and without VirtIO

Each of the five benchmarks was executed simultaneously on an increasing number of virtual machines from 1 to 10 to determine how the performance degrades as the host's load increases for the various benchmarks. Due to KVM's multiple options for storing hard disk images and presenting them to the virtual machine two sets were executed on KVM one with the virtual machines using the VirtIO drivers and another with the virtual disk drive connected as an IDE disk. Both KVM options used copy-on-write image files while Xen used RAW image files.

6.2.1 Ideal lines

Many of the Figures in the next sections include an ideal line which helps indicate how an "ideal" hypervisor should behave when adding additional CPU or memory intensive virtual

machines. The Ideal line assumes minimal or no hypervisor overhead and that once a virtual machine starts it is pinned to a specific processing core.

Since the physical servers in the testing environment contain four processing core the ideal lines show that the results from tests with up to and including four running virtual machines should be equal. This comes from the fact that each virtual machine will have a processing core dedicated solely to itself. Once more than one virtual machine is active on a core processing time is divided equally between all the virtual machines on that core. Additionally, if another virtual machine is activated it will be pinned to the processing core containing the fewest number of active virtual machines. The average performance of all active virtual machines is then calculated based on this division of load and included in the various figures.

6.2.2 Apache Benchmark

The Apache Benchmark is designed to test the performance a given server can provide. It does this by stressing a given server to determine how many requests per second it is able to handle. The version of the benchmark that is included with PTS attempts to execute 500,000 requests to the server 100 requests at a time. It then measures how many request per second the system is able to sustain.

This benchmark was chosen as it gives a good idea how the hypervisor is able to handle increasing I/O stress in terms of CPU and memory usage as the number of virtual machines increases. Figure 6.1 shows how increasing the number of virtual machines affects the performance of the other. According to the chart the Xen PV configuration is least affected by the increasing number of running virtual machines and most closely matches the ideal line. However, as can be seen in Table 6.1 the standard deviation increases significantly for Xen PV when more than a one virtual machine is running on each CPU core. This can indicate that the hypervisor is not necessarily allocating equal amount of execution time to each virtual machine allowing some to perform better than others.

Both KVM configurations behave similarly as additional virtual machines are added; with the VirtIO enabled virtual machines performing slightly better than the other KVM configuration. The constant underperforming configuration, Xen HVM, is interesting in that the results

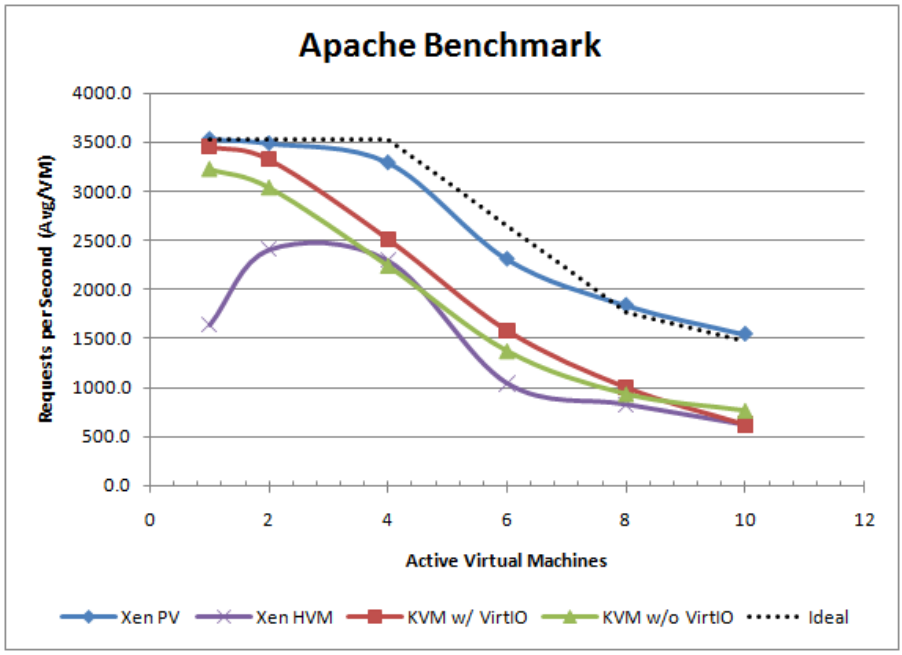

Figure 6.1 Apache Benchmark Results

	Xen PV		Xen HVM		KVM w/ VirtIO		KVM w/o VirtIO	
VMs	Average (Req/Sec)	Standard Deviation	Average (Req/Sec)	Standard Deviation	Average (Req/Sec)	Standard Deviation	Average (Req/Sec)	Standard Deviation
1	3531.0		1639.3		3455.0		3222.0	
2	3488.0	49.5	2409.1	447.6	3323.5	3.5	3033.5	4.9
4	3292.0	101.4	2295.8	159.3	2511.8	308.3	2235.8	113.8
6	2308.7	337.3	1042.8	157.9	1580.5	108.8	1369.5	60.8
8	1839.4	416.0	827.6	184.1	999.9	125.4	931.1	67.3
10	1543.3	445.3	619.0	160.6	621.1	42.9	762.3	98.0

Table 6.1 Apache Benchmark Data: Requests per Second (Average per VM) and Standard Deviation

from a single active virtual machine is worse than with two or four active virtual machines. The same set of tests was run multiple times with similar results each time. This could be cause by some type of issue within PTS or possibly in the way Xen is handling and managing the HVM virtual machines.

This benchmark shows that under the conditions of its test that the paravirtualization within Xen has the capability of performing better but the appearance of higher performance can be offset by the much higher standard deviation within the Xen PV results.

6.2.3 John the Ripper

The John the Ripper benchmark is used to test how the various hypervisors perform under very calculation intensive operations. In this particular PTS test the number of Blowfish calculations per second is measured. Each time the John the Ripper test is executed the PTS will run the benchmark three times and then average those values to get its final result for that run. The PTS result for one virtual machine is then averaged with the other active virtual machines on that hypervisor, if any, running for that test.

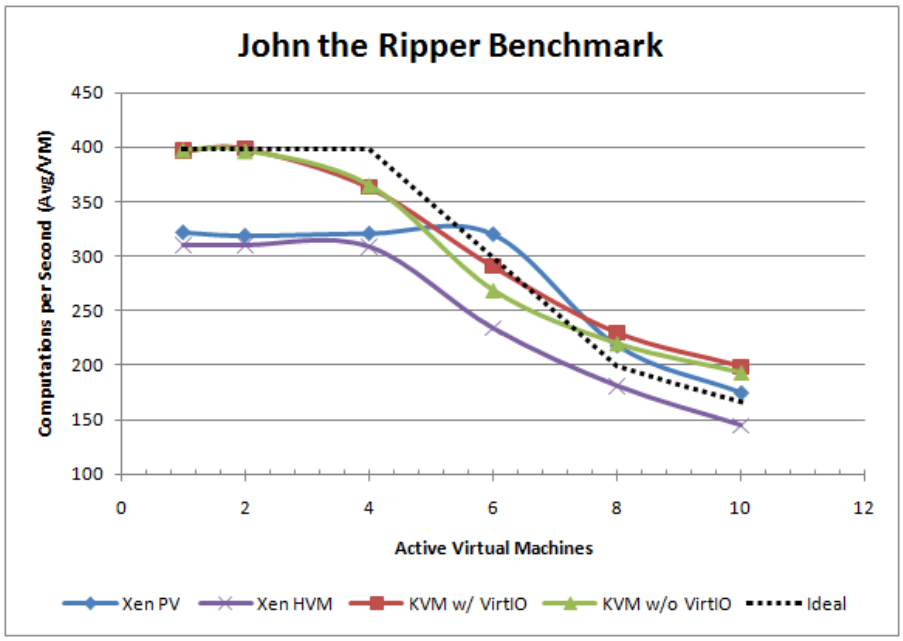

Figure 6.2 John the Ripper Benchmark Results

	Xen PV		Xen HVM		KVM w/ VirtIO		KVM w/o VirtIO	
VMs	Average (BF/Sec)	Standard Deviation	Average (BF/Sec)	Standard Deviation	Average (BF/Sec)	Standard Deviation	Average (BF/Sec)	Standard Deviation
1	322		310		397		398	
2	319	2.8	310	1.4	399	4.2	397	4.2
4	321	0.8	308.5	1.9	363.3	7.9	365.3	23.6
6	320	1.0	233.7	17.4	290.5	40.3	269	14.9
8	218	17.7	180.9	12.0	229.9	28.9	220.3	35.0
10	175	27.9	144.6	13.1	199	35.9	193.5	58.3

Table 6.2 John The Ripper Benchmark Data: Blowfish Computations per Second (Average per VM) and Standard Deviation

The results from John the Ripper test can be seen in Figure 6.2. The ideal line is include and is calculated as specified in section 6.2.1. From Figure 6.2 and Table 6.2 it can be seen that

the KVM configurations tended to have a higher average Blowfish computations per second per virtual machine than Xen in all cases except when six virtual machines were active. The trend in the other configurations make it look like the Xen PV with six virtual machines is an error. However, the standard deviation for this run is very low compared to the others and for verification another PTS run was made in this configuration providing similar results. This bump could be attributed to the way the hypervisor's scheduler is switching between virtual machines or some other anomaly within the PTS test itself.

6.2.4 Unpack

The Unpack benchmark from PTS measures the time required to extract a Linux kernel from a tar.bz2 archive. This benchmark's performance is mainly a factor of disk read/write speed but the CPU is also a factor to uncompress the bz2 archive.

Once again, the Xen PV outperforms the other configurations which was expected since the guest OSs have been optimized for the paravirtualized I/O environment. Xen HVM being the worst performing was expected since Xen HVM creates a loopback device in Dom0 for each HVM virtual machine disk adding additional overhead.

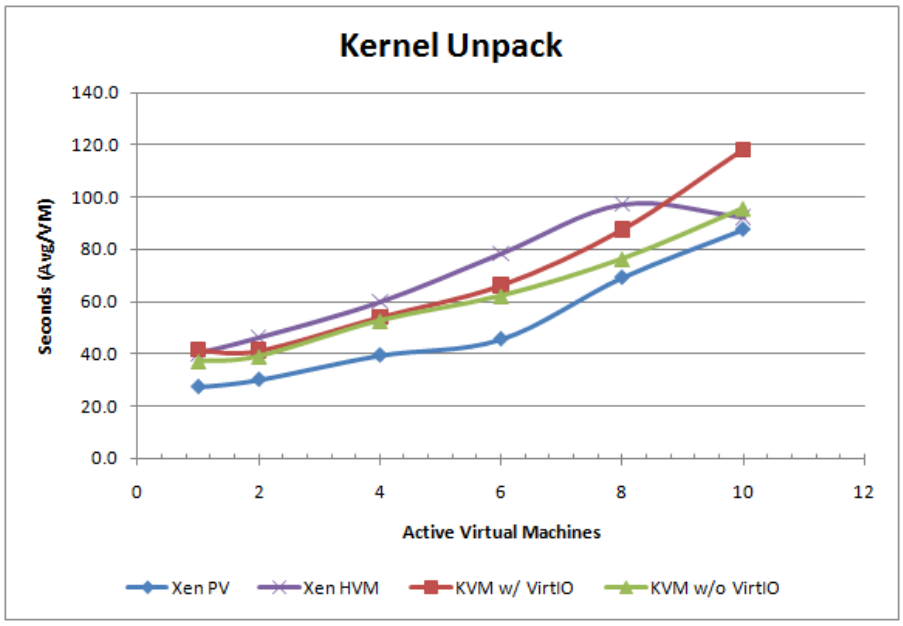

Figure 6.3 Kernel Unpack Benchmark Results

	Xen PV		Xen HVM		KVM w/ VirtIO		KVM w/o VirtIO	
VMs	Average (Seconds)	Standard Deviation	Average (Seconds)	Standard Deviation	Average (Seconds)	Standard Deviation	Average (Seconds)	Standard Deviation
1	27.4		40.2		41.6		37.3	
2	30.2	0.45	46.2	3.02	41.3	7.02	39.2	0.79
4	39.5	2.99	59.9	9.02	54.1	14.0	52.8	3.90
6	45.7	5.30	78.5	23.5	66.3	22.4	62.2	12.4
8	69.3	7.41	97.3	14.3	87.5	26.2	76.4	18.9
10	87.7	12.81	92.4	17.6	118.2	27.1	95.6	18.3

Table 6.3 Kernel Unpack Benchmark Data: Time (Seconds) to Extract (Average per VM) and Standard Deviation

What was not expected was that KVM without VirtIO drivers consistently performed slightly better than KVM with VirtIO drivers. Since the VirtIO drivers provide the virtual machine with a more paravirtualized method of accessing block (disk) and network devices one would expect them to perform better then without the assistance of the paravirtualized interface.

There are a few explainations of this behavior. One is how VirtIO does disk caching. Several disk caching modes exist within KVM (off, write through, and write back) each of which performs differently under various conditions. Another possible explainations is RAW image files vs copy-on-write disk files. Since KVM was using copy-on-write disk files this could affect the virtual disk performance especially on write operations when compared to the Xen RAW image files.

6.2.5 Stream

The Stream benchmark in PTS is built from a program which was designed to measure the sustainable memory bandwidth rather than burst or peak performance of a machine. The Stream benchmark has four operating modes: copy, scale, sum, triad. In this test, only the copy mode was used as the other three tests rely more heavily on the CPU to do some computations on the data being before writing it to memory. This is in contrast to copy which measures transfer rates without doing any additional arithmetic, it instead copies a large array from one location to another. The benchmark specifies the array so that it is larger than the cache of the machine and structured so that data re-use is not possible. [11]

Results from this test are in Figure 6.4 and Table 6.4. For this test instead of an ideal line

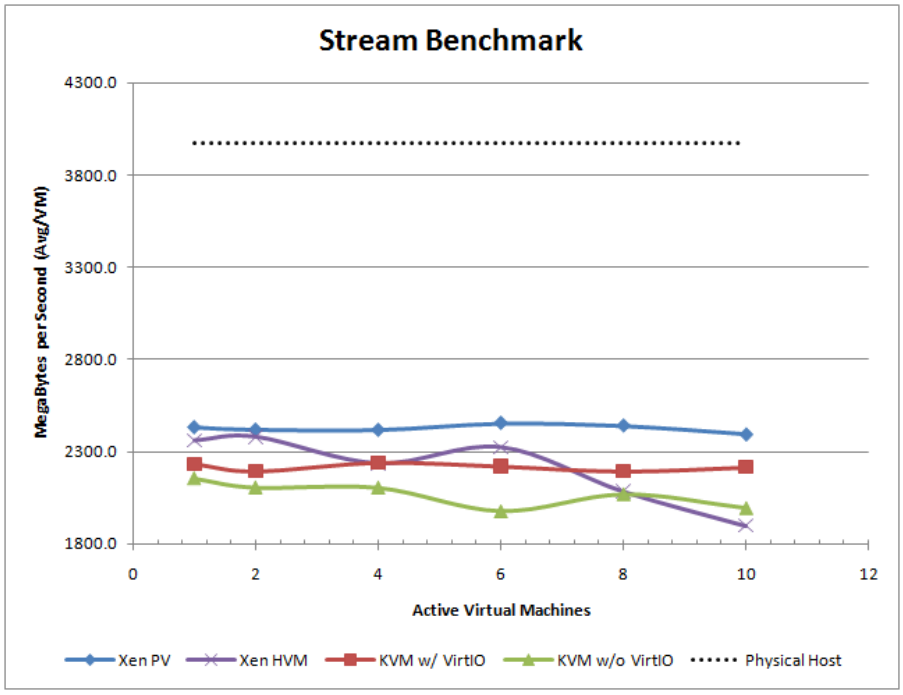

Figure 6.4 Stream Benchmark Results

	Xen PV		Xen HVM		KVM w/ VirtIO		KVM w/o VirtIO	
VMs	Average (MBps)	Standard Deviation	Average (MBps)	Standard Deviation	Average (MBps)	Standard Deviation	Average (MBps)	Standard Deviation
1	2432.0		2359.5		2234		2157.0	
2	2418.8	11.14	2379.7	10.97	2197	16.26	2107.0	25.46
4	2417.5	36.60	2236.7	78.39	2241	5.72	2106.3	21.96
6	2454.5	10.69	2323.9	4.65	2221	9.61	1975.8	111.5
8	2440.6	16.98	2084.5	58.60	2196	50.21	2069.8	38.80
10	2393.2	127.3	1899.0	155.8	2216	9.62	1993.3	118.2

Table 6.4 Stream Benchmark Data: MegaBytes per Second (Average per VM) and Standard Deviation

being calculated, the benchmark was run on the KVM host machine to aid in determining the hypervisor induced overhead. As expected the host machine is able to significantly outperform the virtual machines.

The various Xen and KVM configurations average throughput per virtual machine remained fairly consistent as the number of virtual machines increases. This is especially true when compared to the relative drop off of some of the other PTS tests. This shows that the hypervisor is able to equally distribute memory access resources between the active virtual machines on the system.

From Figure 6.4 it's easy to see that the throughput of the data bus is not the issue affecting the performance. Instead, the reduced throughput of the virtual machines is most likely caused by hypervisor related overhead. Each time the virtual machine attempts to access memory the hypervisor must translate that access to a physical memory address on the host adding some delay. Additionally, the hypervisor must constantly switch between which virtual machine to serve at a given time constantly refreshing the physical host's cache.

6.2.6 OpenSSL

OpenSSL is another computation benchmark within PTS. This particular test uses the open source OpenSSL toolkit to measure the performance of signing and verifying using 4096bit RSA operations. The results of this test are in Figure 6.5 and Table 6.5. KVM and Xen HVM stay fairly consistent as the number of virtual machines increases with a minimal decrease in performance. Additionally, the low standard deviation indicates that the hypervisors are able to fairly allocated resources between the active virtual machines.

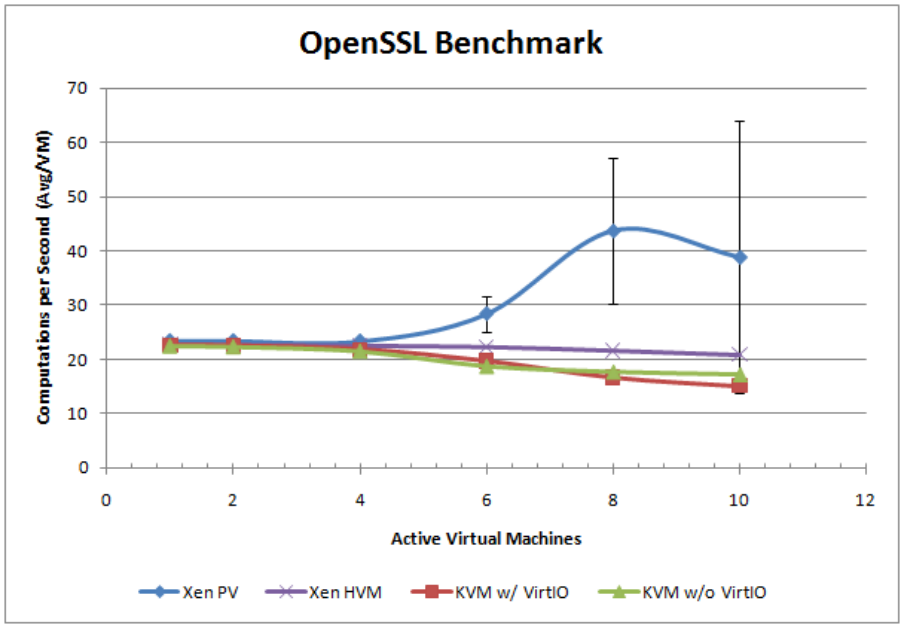

Figure 6.5 OpenSSL Benchmark Results

According to the data the Xen PV performance increases as multiple virtual machines are assigned to each processing core. However, the standard deviation also increases significantly

VMs	Xen PV		Xen HVM		KVM w/ VirtIO		KVM w/o VirtIO	
	Average (RSA/S.)	Standard Deviation	Average (RSA/S.)	Standard Deviation	Average (RSA/S.)	Standard Deviation	Average (RSA/S.)	Standard Deviation
1	23.4		22.68		22.6		22.43	
2	23.4	0.00	22.64	0.014	22.5	0.049	22.31	0.177
4	23.3325	0.05	22.48	0.067	21.8	0.293	21.48	0.178
6	28.39	3.27	22.26	0.102	19.7	1.346	18.75	1.008
8	43.68143	13.36	21.62	0.351	16.7	1.516	17.72	0.971
10	38.805	25.09	20.87	0.258	15.1	2.103	17.25	1.487

Table 6.5 OpenSSL Benchmark Data: RSA Computations per Second (Average per VM) and Standard Deviation

for these as well to the point where it is over 50% of the average computations per virtual machine. This behavior is not practical and required further investigation.

6.2.6.1 OpenSSL PTS Details

The OpenSSL test will run and count how many times it can do 4096 bit RSA operations in 10 seconds. Each time the OpenSSL PTS benchmark is executed the suite will run this OpenSSL test four times, average the results and present the average as the benchmark's result for that virtual machine. This benchmark result is then averaged with the benchmark results from all other active virtual machines to get the average per virtual machine included in the graph.

6.2.6.2 Erroneous Results

Looking more closely at the Xen PV OpenSSL test results from when greater than four virtual machines are active there are several results which are not practical and skew the results.

Typical Results	**Erroneous Results**
124 4096 bit private RSA's in 5.62s	125 4096 bit private RSA's in 0.00s
125 4096 bit private RSA's in 5.53s	123 4096 bit private RSA's in 0.00s
125 4096 bit private RSA's in 5.04s	125 4096 bit private RSA's in 0.59s
114 4096 bit private RSA's in 5.34s	125 4096 bit private RSA's in 0.80s

Table 6.6 Sample typical and erroneous results from the OpenSSL tests

In Table 6.6 the first two results under Erroneous Results are obviously not valid as it would be impossible for the computations to be completed in 0.00 seconds. Even the following two

are difficult to believe as they calculate out to be over 150 RSA operations per second, still significantly high campared to the typical results..

6.2.6.3 Further Investigation

Upon further investigation it was determined that under certain circumstances Linux virtual machines can have clock and time synchronization issues on various hypervisors. There was one specific example explained in [14] where when issuing the "date" command multiple times in a row running "date;date;date;date;date;date;date" the returned time would switch between 40 seconds and 48 seconds every other execution. To help mitigate this particular problem a knowledge base page [9] from VMware, another virtualization provider, was used which provided kernel parameters for several flavors of Linux to "achieve best timekeeping results".

As a test, the provided Kernel parameter for RHEL 5.3, "divider=10 clocksource=acpi_pm", was added to the Xen PV virtual machines and the OpenSSL benchmark was re-run with 6, 8, and 10 active virtual machines.

6.2.6.4 OpenSSL Modified Results

Table 6.7 contains the results of the new and original Xen PV results alongside the KVM with VirtIO values for comparison. The standard deviation for the modified Xen PV is significantly better for eight and ten active virtual machine but is still not as good as the KVM test as illustrated in Figure 6.6.

	Xen PV		Xen PV with Parameter		KVM w/ VirtIO	
VMs	Average (RSA/Sec)	Standard Deviation	Average (RSA/Sec)	Standard Deviation	Average (RSA/Sec)	Standard Deviation
6	28.39	3.27	31.36	5.02	19.7	1.346
8	43.68143	13.36	22.275	5.68	16.7	1.516
10	38.805	25.09	22.69	4.29	15.1	2.103

Table 6.7 OpenSSL Benchmark Data: with and without Kernel parameter

6.3 Simulated Users

While high performance and throughput is important in some applications many times, including in Xen Worlds, one is willing to sacrifice some of that performance if virtual machine

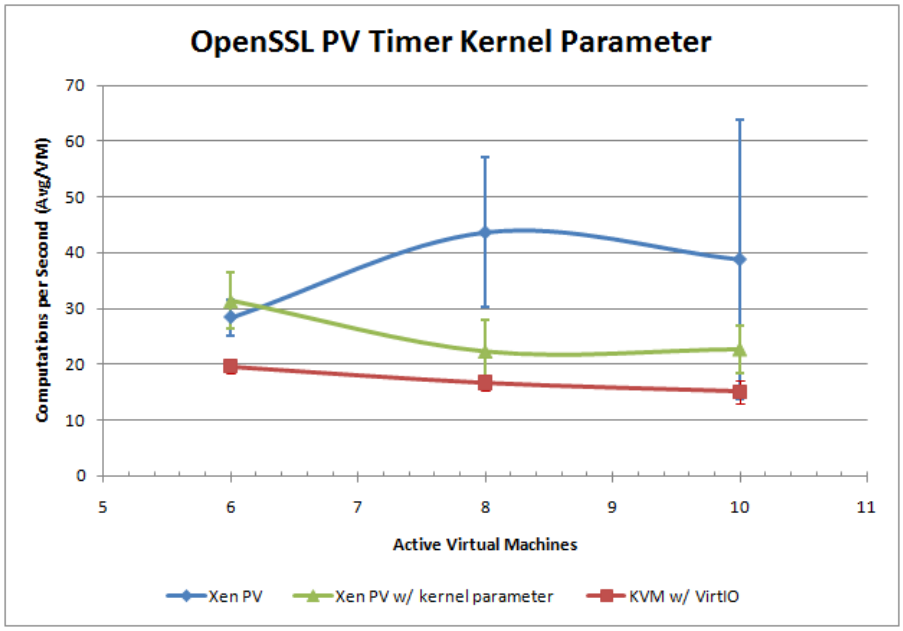

Figure 6.6 OpenSSL Benchmark with and without Kernel parameter

density can be increased. This was one of the primary goals driving the original development of Xen Worlds [2], to optimize hardware utilization without making a noticeable effect on the user experience. The original developers made use of Expect scripts in an effort to gauge the environment's responsiveness under regular user operating conditions. Similar Expect scripts were used again to compare the responsiveness of KVM and Xen from the user's perspective.

Ten Expect scripts, based on the original scripts used in [2], were created which simulate how a typical user might interact with the Xen Worlds environment while completing a lab assignment.

The Expect scripts connect, via SSH, to the lab and navigate the Pdmenu menu interface to select a virtual machine, just as a regular user would. Each of the ten scripts then perform a unique combination of the following commands. In addition each script will, at least once, return to the Pdmenu and select a different virtual machine on which to continue its execution. Each of the Expect scripts include a sub-set of these commands:

- List a directory contents
- Change the working directory

- Use "cat" view the /etc/passwd file
- Use "ps aux" to list current system processes
- Use "netstat" to print network connection
- Run a string reversal program
- Run a "Hello World" program

As in [2] the parameter "*set send_human .1 .3 1 .05 2*" was used within each Expect script which, according to the Expect manual page, will adjust the speed of the text input to a rate similar to a human typing.

6.3.1 Goal of the Expect Scripts

The PTS benchmarks are a valid test to determine how virtual machines performance and throughput varies as the number of virtual machines performing the same computationally or I/O intensive task changes. In a virtual environment it is very rare for virtual machines to be performing identical tasks trying to max out the physical host. This is especially true in with Xen Worlds. It is very rare that deployed User Worlds will be actively and continuously engaged in intensive tasks.

The goal of the Expect Scripts is to gauge how well the two hypervisors are able to isolate one virtual machine from another. This is not isolation in the traditional sense, network and security, instead performance isolation relating to how the operations of one virtual machine affect the performance and responsiveness of others running on the same host. For example, if a user is running John the Ripper to crack passwords on one virtual machine, Expect scripts measure how well the hypervisor is able to minimize it's affect on the responsiveness of another user's virtual machine. This becomes especially important as the number of active Worlds on each physical host increases.

6.3.2 The Test Environment

To carry out the tests Xen User Worlds, each consisting of a set of three virtual machines networked together, were deployed on both Xen and KVM for an varying number of simulated users. This was done in the same way a Xen Worlds assignment would be deployed to a group of users. Again, Xen's virtual machines are running RHEL 5.3 and KVM's virtual machines are running RHEL 6. The virtual machines on both hypervisors this time are only alloacted 128 MB of RAM, the typical amount for the Xen Worlds environment. The results are determined by the amount of time it takes the Expect script to finish executing. Each test was carried out 10 times before the number of User Worlds was changed for the next set of tests.

6.3.3 Concurrent Users Test

For this concurrent users test the number of active users ranges from 1 to 10 results of which can be found in Table 6.8. Ideally, increasing the number of User Worlds should have minimal effect on the responsiveness of the virtual machines and thus the execution time of the Expect scripts. From Figure 6.7 it can be seen that both Xen implementations had some issues. In several of the tests, especially as the number of users increases, some executions of the Expect scripts would timeout after selecting a new virtual machine at the Pdmenu and waiting for that virtual machine to respond with a prompt. In order to reduce the influence these timeouts have on the graph, an "adjusted" plot was added for both Xen PV and Xen HVM which remove any times greater than 5 minutes from the averages.

	KVM		Xen PV		Xen HVM		Xen PV	Xen HVM
User Worlds	Average (Sec)	Standard Deviation	Average (Sec)	Standard Deviation	Average (Sec)	Standard Deviation	Adjusted Average	Adjusted Average
1	114.598	2.797	124.354	2.557	135.388	5.390	124.354	135.388
2	118.833	4.776	125.473	4.721	135.796	3.521	125.473	135.796
3	117.435	3.662	135.385	45.022	135.875	4.306	127.239	135.875
4	116.843	3.704	211.983	161.059	137.464	4.436	142.027	137.464
5	116.717	3.538	181.530	134.810	139.485	4.445	136.121	139.485
6	116.731	3.419	284.131	197.955	154.179	82.039	161.683	143.604
7	116.474	4.000	149.307	19.922	152.799	5.645	149.307	152.799
8	117.189	4.122	182.246	106.437	216.070	132.557	159.020	168.255
9	116.839	3.833	197.552	104.064	230.527	93.714	157.216	201.812
10	117.529	3.735	266.339	168.129	528.007	233.679	175.798	224.272

Table 6.8 Simulated Concurrent User Responsiveness Data: Average execution time (sec) and Standard Deviation

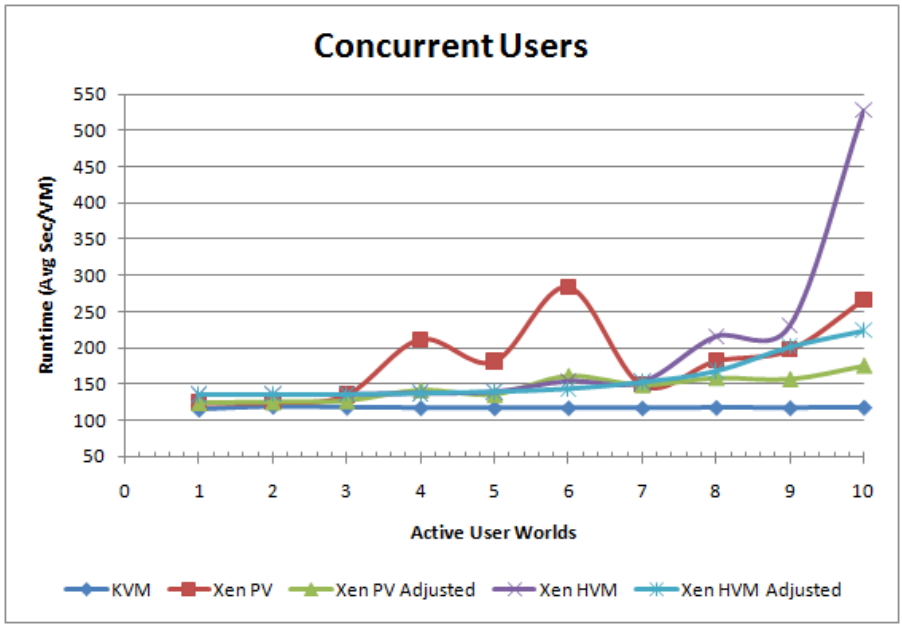

Figure 6.7 Simulated Concurrent User Responsiveness

Even when compared to the adjusted averages KVM has a significant advantage over Xen in guest performance isolation experiencing little slowdown as the number of active users increased, indicating that KVM has a better scaling potential in a Xen Worlds environment.

6.3.4 Single User with varying John the Ripper

Given the nature of the Xen Worlds assignments, there are situations when a user may want to occasionally run computationally intensive applications, like John the Ripper, on their virtual machine. This test measures the responsiveness of a single User World, using the same Expect script, as before in the event that other users are concurrently running John the Ripper. Figure 6.8 shows how increasing the number of virtual machines running John the Ripper affects other users. Once again KVM stays fairly consistent when compared to Xen with execution time increasing only a few seconds as the number of active John the Ripper virtual machines increases. The data in Table 6.9 shows that the standard deviation for all of the tests, both Xen and KVM, stays fairly consistent even as the computational stress is increased.

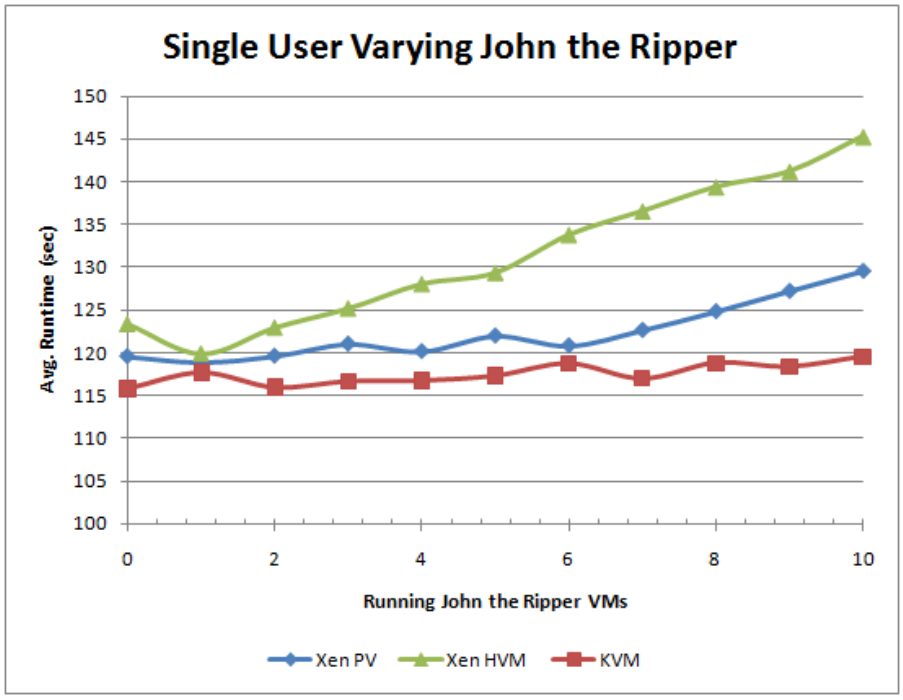

Figure 6.8 Single Simulated User, varying John the Ripper

	KVM		Xen PV		Xen HVM	
JTR VMs	Average (Sec)	Standard Deviation	Average (Sec)	Standard Deviation	Average (Sec)	Standard Deviation
0	115.819	3.806	119.589	2.955	123.388	5.390
1	117.668	4.184	118.924	3.764	119.919	2.768
2	115.957	2.660	119.635	3.699	122.977	4.653
3	116.665	3.141	121.051	2.060	125.205	3.254
4	116.740	3.646	120.180	3.561	128.071	2.756
5	117.305	2.999	122.005	2.690	129.383	3.274
6	118.710	3.459	120.848	1.633	133.816	3.614
7	116.977	3.635	122.667	3.203	136.580	4.620
8	118.793	3.569	124.871	3.148	139.398	4.278
9	118.358	3.121	127.261	3.421	141.229	3.299
10	119.527	3.826	129.610	2.428	145.294	4.002

Table 6.9 Single Simulated User with varying John the Ripper: Average exection time (sec)

6.3.5 Multi User consistent John the Ripper

In a variation of the previous tests, a constant four virtual machines running John the Ripper were created, one for each processing core, and the number of User Worlds was varied from 1 to 10. The goal of this is to see how well the hypervisor performed when there is an active load on each processing core. As indicated in Table 6.10 and Figure 6.9 KVM is still able to stay fairly consistent compared to Xen. Once again Xen PV experienced similar timeout issues and the adjusted values in the table and figure have any times over 5 minutes removed from their averages.

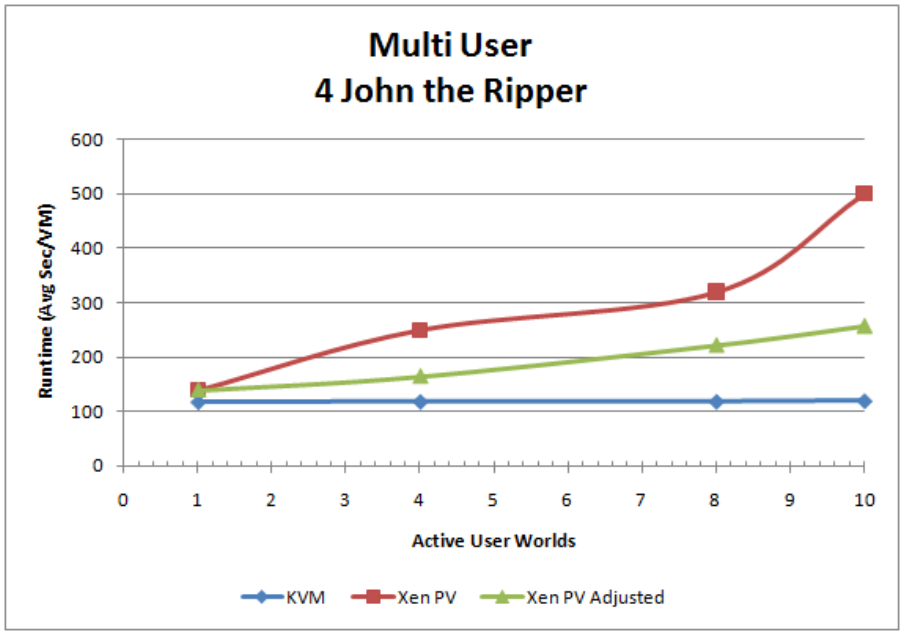

Figure 6.9 Consistent (4) John the Ripper, varying Simulated Users

	KVM		Xen PV		
User Worlds	Average (Sec)	Standard Deviation	Average (Sec)	Standard Deviation	Adjusted Average
1	117.514	3.011	138.910	2.974	138.910
4	118.363	3.164	250.039	179.139	164.424
8	118.418	3.190	319.671	160.892	221.983
10	119.500	3.462	500.042	209.479	257.531

Table 6.10 Consistent (4) John the Ripper, varying Simulated Users: Average exection time (sec)

6.3.6 KVM: Increasing John the Ripper

Given how well KVM was able to maintain relatively consistent performance in the previous tests as the number of User Worlds and John the Ripper virtual machines increased another set test was created to see how well it continued to perform. This test, only done on KVM, varies the number of User Worlds from 1 to 10 and also varies the number of John the Ripper virtual machines from 0 to 12. The goal of this is to determine at what point KVM will no longer be able to handle the load and experience a drastic decrease in virtual machine responsiveness and isolation preformance.

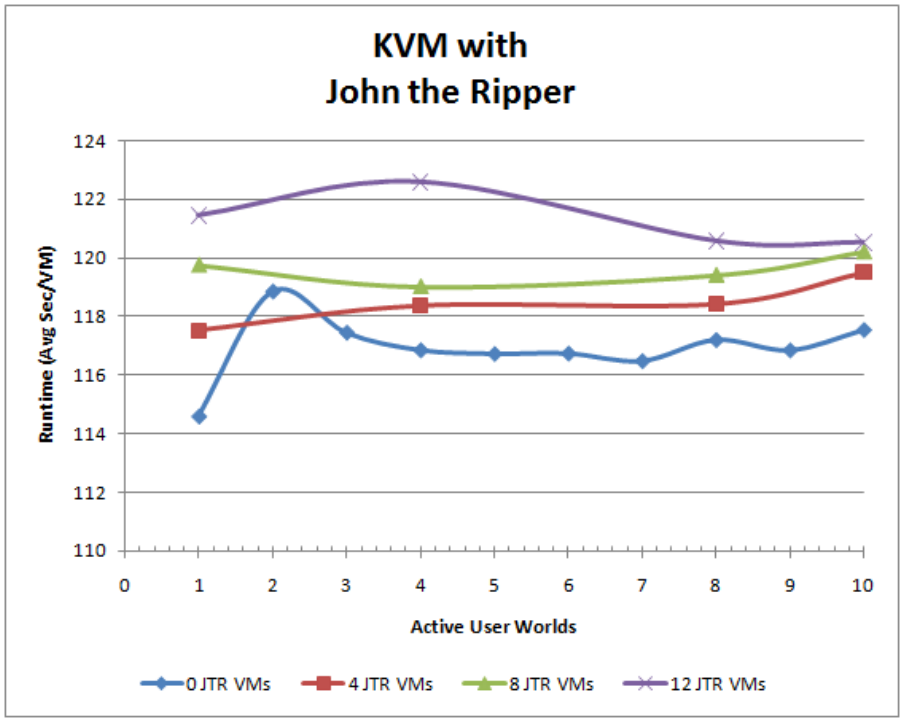

Figure 6.10 KVM only: varying Simulated Users and varying John the Ripper

	No John the Ripper		4 John the Ripper		8 John the Ripper		12 John the Ripper	
User Worlds	Average (Sec)	Standard Deviation	Average (Sec)	Standard Deviation	Average (Sec)	Standard Deviation	Average (Sec)	Standard Deviation
1	114.598	2.797	117.514	3.011	119.752	2.760	121.443	4.153
4	116.843	3.704	118.363	3.164	119.011	3.307	122.576	3.092
8	117.189	4.122	118.418	3.190	119.408	3.659	120.572	3.319
10	117.529	3.735	119.500	3.462	120.207	3.432	120.522	3.858

Table 6.11 KVM only: varying Simulated Users and varying John the Ripper: Average execution time (sec)

The data in Table 6.11 and Figure 6.10 shows KVM is still able to stay fairly consistent as the load increases. As expected, the data points primarily shift upward a few seconds each time an additional four John the Ripper virtual machines are added. Even when 12 John the Ripper and 10 user worlds are active for a total of 42 running virtual machines KVM is able to maintain very good relative performance and a low standard deviation.

6.4 Results Conclusion

The test conducted in this chapter shows that both KVM and Xen have different performance benefits in different situations. The PTS benchmarks give the indication that Xen PV tends to outperform KVM in disk and I/O related scenarios. This would be expected in the paravirtualized environment since guest operating systems are modified to work specifically in such an environment and in the modification process can be better optimized for the environment. KVM tended to perform better in computationally intensive operations like the John-the-Ripper and OpenSSL PTS tests. In these tests KVM was also able to keep the standard deviation relatively low.

The scenarios where KVM undoubtedly excelled were in the simulated user tests. The execution of the Expect scripts present KVM as a clear leader in virtual machine performance isolation especially in a Xen Worlds like environment. In almost all of the scenarios KVM is able to maintain consistent execution times supported by the relatively low and consistent standard deviations. This is most likely due to the method KVM uses to manage its virtualization by making use of very mature and time tested components of the standard Linux kernel for components such as memory management and scheduling.

CHAPTER 7. SPECIAL XEN WORLDS CONSIDERATIONS

Hypervisor and end user performance is not the only aspect to consider when evaluating a virtualization platform for a lab environment. Simplicity of deploying and managing labs is also important especially as class size and lab complexity grows. In this chapter a few performance related changes to the Xen Worlds Middleware that were done during the Xen to KVM conversion will be analyzed.

7.1 Disk Utilization

One of the most desired feature additions made available from the migration from Xen to KVM was copy-on-write image files. This has greatly reduced the amount of disk space required to store virtual disk images. As stated in section 4.2 copy-on-write images allow for multiple overlay images to be created using a single base image. This allows for an overlay image to be created for each deployed virtual machine instead of requiring a full copy.

Depending on the lab, a Xen hypervisor based virtual machine image file ranges from 1.5GB to 3GB. Figure 7.1 illustrates how as the number of virtual machines increase so does the required disk space. The figure assumes a 1.5GB base image for both Xen and KVM. With KVM using copy-on-write images disk space required for each virtual machine will vary depending on the assignment and how much writing to the disk done by the individual user. The KVM: Fresh Deployment line indicates disk usage immediately after the lab is deployed and configured. At this point each virtual machine's overlay image is about 19MB (this includes a snapshot of the virtual machine in its current state). Depending on the characteristics of the assignment the overlay images will likely increase as the user performs the assignment. In Figure 7.1 KVM: 200MB User Data is shown to give an indication of what disk usage would

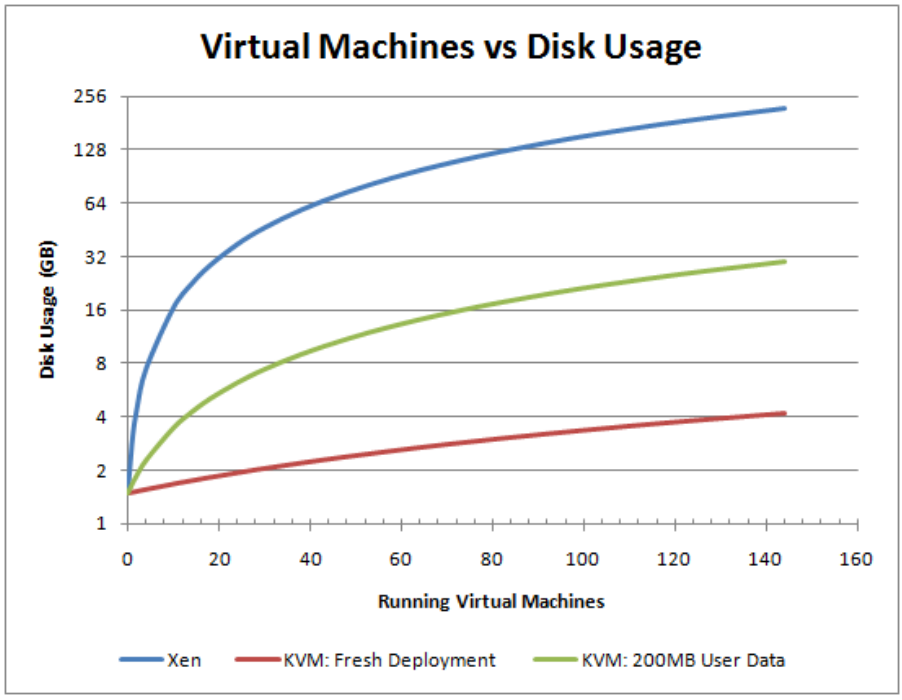

Figure 7.1 Disk usage requirements KVM copy-on-write vs Xen full copy

be if 200MB of user data is added to each of the overlay images. Since Xen does a full of copy of the base image its disk utilization doesn't increase as the virtual machines are used.

Given that some of the current labs require three virtual machines for each user, 120 deployed virtual machines at once is not unreasonable. In some cases multiple assignments could be deployed at once with only one being active further increasing the required disk space.

7.2 Deployment Time

In the Xen hypervisor implementation of the Xen Worlds Middleware it takes almost 2 hours to deploy and configure a lab with three virtual machines to 10 users. Given this it can take all day to deploy a lab to an entire class. This time can be attributed to several factors including:

1. Each virtual machine requiring a full copy of the base image.

2. Starting each virtual machine during deployment in order to make lab and user specific customizations via an Expect script.

3. Consecutive deployment and configuration of each virtual machine.

4. Intentional delays to help prevent race conditions on the Xen hypervisor.

Each of the above factors was analyzed and at least partially mitigated when migrating to KVM to the point where deploying the same three virtual machine lab to 10 users now takes about 5 and a half minutes compared to the 2 hours with the Xen implementation. The following approaches were used to reduce the deployment and configuration time commitment.

1. Copy-on-write images: almost instantaneous creation of an overlay image.

2. Direct modification of the virtual machine image and using the rc.local file within Red Hat to configure the virtual machine on its first boot instead of during deployment.

3. Process-based parallelism: multiple processes are spawned to deploy up to 6 virtual machines at once.

4. Removal or reduction of the intentional delays within the middleware.

7.3 Per-User Customization of Pdmenu

The Xen based implementation required the administrator to manually modify the Pdmenu global configuration file each time a new assignment is deployed and when an assignment is removed. This can be troublesome especially since it is a global file and any changes affect all users. It can also result in added confusion because if, for example, assignment A is deployed to group 1 and assignment B to group 2 the Pdmenus will display options for both assignments to both groups. In this case, depending on the group, one of the options will return an error when attempting to access an invalid assignment.

In the KVM implementation the Pdmenu is dynamically generated for each user based on the labs that are currently deployed to that specific user. The administrator no longer needs to manually modify the Pdmenu files, greatly simplifying management.

CHAPTER 8. CONCLUSION & FUTURE WORK

While Xen may be a more well known hypervisor from the evaluations performed here KVM certainly has what it takes to give it a run for its money. This is especially true for a lab environment such as Xen Worlds. KVM is able to provide far superior performance isolation between virtual machines and does not require modifications to the guest operating system.

In order to conduct these test the Xen Worlds Middleware and labs were migrated or rebuilt so they could be used on the KVM platform to provide an equivalent ground for evaluation. Two classes of tests were performed throughput performance and simulated users. Paravirtualized Xen tends to outperform KVM in I/O type tests. The opposite is true in the simulated user responsiveness tests where KVM is able to better isolate CPU intensive virtual machines mitigating their affect on other virtual machines. This is a very important factor in the Xen Worlds lab environment.

Through the process of migrating the Xen Worlds Middleware to KVM, the Middleware was also enhanced to make use of KVM's expanded feature set and capabilities such as copy-on-write images and snapshots. This has significantly reduced the time required to deploy labs within the environment simplifying management.

There are still several enhancements that could be performed to both expand on the evaluation of the KVM hypervisor for use in a lab environment and to expand the feature set of Xen Worlds running on KVM. KVM has a number of features such as Kernel Samepage Merging (KSM) which can be very useful in a lab environment especially on host systems with more constrained memory. Additionally, KVM can be used with SELinux enabled on the host which can further help isolate virtual machines from each other and help prevent tampering, accidential or intentional, with virtual guests and their image files.

On the Xen Worlds features side, additional labs could be developed possible even a vir-

tualization lab. Limited testing of runing the Xen hypervisor within KVM was done by the author to determine that it is technically possibly to run a Xen Dom0 as a KVM guest. Further lab development could be done using the Windows operating system, although this would also require developing a method for accessing the Windows console as the current Pdmenu interface would not suffice.

Further work could also be done on implementing a cluster ability to Xen Worlds. This could allow for easily expanding the environment by network booting additional host machines and utilizing KVM's live migration ability to actively load balance between the various hosts.

BIBLIOGRAPHY

[1] Benjamin Anderson. Xen worlds project. http://home.eng.iastate.edu/~hawklan/xw-index.html, 2009.

[2] Benjamin Anderson. Xen worlds: Creating a virtual laboratory environment for use in education. Master's thesis, Iowa State University, 2010.

[3] Red hat bugzilla - bug 486353. https://bugzilla.redhat.com/show_bug.cgi?id=486353, 2009.

[4] Intel Corporation. Enhanced virtualization on intel architecture-based servers. Technical report, Intel Solutions, March 2005.

[5] Todd Deshane, Zachary Shepherd, Jeanna N. Matthews, Muli Ben-Yehuda, Amit Shah, and Balaji Rao. Quantitative comparison of xen and kvm. *Xen Summit,*, June 2008.

[6] Red Hat Inc. *Red Hat Enterprise Linux 6 Virtualization Guide*, 2008,2009,2010.

[7] Red Hat Inc. Kvm - kernel based virtual machine. Technical report, Red Hat Inc., 2009.

[8] VMware Inc. Understanding full virtulization, paravirtualization, and hardware assist. Technical report, VMware Inc., 2007.

[9] VMware Inc. Timekeeping best practices for linux guests. http://kb.vmware.com/selfservice/microsites/search.do?language=en_US&cmd=displayKC&externalId=1006427, 2011.

[10] VMware Inc. Transform your business with virtualization. http://www.vmware.com/virtualization/history.html, 2011.

[11] John D. McCalpin. Stream: Sustainable memory bandwidth in high performance computers. http://www.cs.virginia.edu/stream/.

[12] Phoronix Media. Phoronix test suite. http://www.phoronix-test-suite.com, 2008 - 2011.

[13] Redhat enterprise linux 6 xen 4.0 tutorial. http://wiki.xen.org/xenwiki/RHEL6Xen4Tutorial, 2005-2011.

[14] Weird fluctuating time on a xen linux guest. http://serverfault.com/questions/116637/weird-fluctuating-time-on-a-xen-linux-guest.

[15] Amit Singh. An introduction to virtualization. http://www.kernelthread.com/publications/virtualization/, 2004.

[16] What is xen hypervisor? http://www.xen.org/files/Marketing/WhatisXen.pdf, 2005-2011.

[17] Why xen? http://www.xen.org/files/Marketing/WhyXen.pdf, 2005-2011.